Image-based Tree Variations

Abstract

The automatic generation of realistic vegetation closely reproducing the appearance of specific plant species is still a challenging topic in computer graphics. In this paper we present a new approach to generate new tree models from a small collection of frontal RGBA images of trees. The new models are represented either as single billboards (suitable for still image generation in areas such as architecture rendering) or as billboard clouds (providing parallax effects in interactive applications). Key ingredients of our method include the synthesis of new contours through convex combinations of exemplar countours, the automatic segmentation into crown/trunk classes, and the transfer of RGBA color from the exemplar images to the synthetic target. We also describe a fully-automatic approach to convert a single tree image into a billboard cloud by extracting superpixels and distributing them inside a silhouette-defined 3D volume. Our algorithm allows for the automatic generation of an arbitrary number of tree variations from minimal input, and thus provides a fast solution to add vegetation variety in outdoor scenes.

CCS Concepts

•***Computing methodologies*** → *Image-based rendering;*

1. Introduction

Vegetation (trees, bushes and grass) is an essential part of natural outdoor scenes. Many different plant representations have been proposed in the literature, including polygonal-based, image-based [BCF*05], point-based [DCSD02], volume-based [DN04], relief mapping-based [ACA16], and procedural [ACV*14] approaches. Here we focus exclusively on image-based representations of trees.

Billboards are the simplest tree representation and as such they are extensively used today in many applications including architecture rendering and video games. A billboard consists of a single texture-mapped quad that is rotated around its vertical axis so that it always faces the camera, thus exploiting the apparent axial symmetry of trees. Billboard textures are RGBA, where the alpha (opacity) channel segments foreground (tree) from background pixels, allowing non-tree parts of the texture to be easily discarded.

Billboards suffer from some well-known limitations. They do not support top views of the trees, since the image only shows a side view, and the billboard is allowed to rotate only around its vertical axis. Since billboards are planar, shading effects are limited, even when equipped with normal maps. Cast shadows correspond to that of the tree silhouette and these shadows are received as if the trees were planar. Concerning real-time rendering, billboards do not support view-motion parallax, and the fact that they always show the same side of the tree might become apparent e.g. when the camera rotates around the tree.

Despite the limitations above, the simplicity, speed and compactness of billboards, the realism provided by actual photographs of trees, and the availability of large collections of ready-to-use tree textures make tree billboards ubiquitous in many CG applications.

In this paper we explore different strategies for the automatic generation of tree variations from a collection of RGBA images (this collection will be referred to as the tree library). We discuss the creation both of single-billboard trees, suitable for still-image rendering, and billboard clouds, that can be used in interactive applications and support parallax effects and arbitrary view directions.

We consider two main usage scenarios. The first one is oriented towards the *authoring* of new tree images. Given a user-defined subset of exemplars from the tree library, the algorithm creates plausible trees through random combinations of the chosen exemplars. These variations can be used to enrich existing plant libraries. The second scenario aims at avoiding the perception of repeated copies of trees. If the same tree image is instanced multiple times, users will easily detect the repeated copies, resulting in poor, overly repetitive vegetation. Straightforward per-instance variations (e.g. scaling or HSL color coding perturbations) do not suffice to overcome the impressive pattern-matching ability of the human visual system. Given an input tree image, the algorithm creates an arbitrary number of variations of the tree image by perturbing its shape and appearance using features extracted from the library.

A key ingredient of our approach is the combination of multiple tree shapes and color textures so that the resulting images have a realistic tree appearance. Regarding the overall tree shape, we encode tree contours by a fixed-length high-dimensional vector that enables high-quality morphing between two or more tree shapes.

Concerning the tree appearance, we transfer color (and opacity values) between tree images through the use of Mean Value Coor-

dinates [HF06] computed over simplified contours. We also present a fast and efficient method to segment tree images into crown/trunk pixels, so that their respective contours and appearances are combined and transferred properly.

The main contributions of the paper are: (a) a completely-automatic pipeline for creating single-image tree variations from existing RGBA images of trees; (b) a robust method to extract the overall shape of a tree, including the automatic trunk segmentation; (c) the encoding of the overall external contour of a tree as a fixed-length high-dimensional vector; arbitrary contours are supported, not being limited to convex nor star-shaped polygons; highly fractal contours are supported; (d) the synthesis of random contours through convex combinations of contours from the library; (e) methods for transferring color and opacity values from one or more source exemplars to a target image; the use of Mean Value Coordinates along with pinned contour points found through trunk segmentation avoids excessive area distortion during color transfer; (f) the automatic conversion of a single tree image into a billboard cloud; we extract compact superpixels with foliage-like boundaries, and distribute them inside a plausible 3D reconstruction of the tree.

The rest of the paper is organized as follows. Section 2 reviews relevant previous work on tree creation through procedural, inverse procedural, and contour-based techniques. Section 3 provides an overview of our preprocessing and synthesis algorithms, which are further developed in Sections 4, 5 and 6. Results with a large collection of tree exemplars are presented in Section 7. Conclusions are presented in Section 8.

2. Previous work

2.1. Procedural generation

The problem of generating plausible vegetation has received constant attention in computer graphics research. The first techniques used to model trees were based on procedural generation due to the lack of proper scanning devices. One set of techniques introduced by Lindenmayer [Lin68] exploited the capabilities of formal languages and managed to imitate plant development. L-systems have been widely used for modeling all types of plants and have been extended to support most of its peculiarities. These extensions include the development of the plant over time [PHM93], the interaction with the environment [MP96], the expression of plant attributes based on their spatial location [PMKL01], and many others [PL96].

A different possibility is to exploit the competition for resources (sunlight, space) by different branches of a tree which seems to be critical for the general shape of temperate-climate trees. The space colonization algorithm presented by Runions et al [RLP07] uses this fact. Palubicki et al. [PHL*] extended this algorithm using a signaling mechanism to mimic different types of growth. In [XM15] Xu and Mould improved the performance of tree generation algorithms based on space competition by using pregenerated guiding vector fields and Yao graphs. Kohek and Strnad [KS15] on the other hand used competition for incoming light as the main competition resource.

Wind and surrounding space also influence a tree's growth. Pirk et al. [PSK*12] presented a technique that made it possible for content creators to change a tree's position inside a scene and see its shape adapt to changes in light distribution and the occupation of surrounding space. In [PNH*14] it was extended to include the effect of wind.

2.2. Tree generation from contours

Generating a tree from its contour is already possible applying the space colonization algorithm [RLP07], but there are other contributions that compute a tree from its silhouette. It is also possible to provide the overall crown shape using a gesture based system that guides the resulting tree's growth [LRBP12], reducing modeling time while maintaining the artist's ability to obtain the desired result. Okabe et al [OOI05] were able to generate 3D tree models from 2D sketches by taking the assumption that trees spread their branches far apart from each other. Wither et al. [WBCG09] made use of successive silhouettes sketched at different zoom levels to create a 3D tree.

Other techniques generate trees from a single photograph. These algorithms extract the silhouette of the crown and use it to generate the crown and its foliage. In [TFX*08] the user draws strokes in the photograph to identify the crown and the branches. The crown is segmented and the visible branches are converted to 3D using the approach proposed in [OOI05]. The initial skeleton is extended into the crown by iteratively substituting an existing branch by a subtree from a database. Guenard et al. [GMBC13], on the other hand, extract the foliage and compute a vector field from the segmented shape that is used to obtain a skeleton. This base skeleton is populated by leaves and branchlets via an L-system. For dense trees it is possible to extract more compact representations. Argudo et al [ACA16] present a complete pipeline for synthesizing and rendering detailed trees. A rough estimate of the crown shape is created by solving a thin-plate energy minimization problem. Then detail is added through a simplified shape-from-shading approach.

2.3. Inverse procedural modeling

One way of generating variations from a single tree could be to identify the input parameters of a given procedural modeling algorithm, then alter those parameters slightly. Despite the difficulty of predicting the outcome produced by procedural modeling grammars, it is possible to control the process and produce the desired output. One possibility is to let the user explore the space of all possible models while guiding this search [TGY*09]. In order to help users explore the space of possible models it is interesting to provide a relatively small set of parameters that control the generation. These may be provided by the procedural algorithm designer but they may also be obtained semi-automatically. In [YAMK15] Yumer et al. use autoencoder networks to find a lower dimensional space that can be used as the set of parameters for users to play with. A weak point of these systems is that the resulting trees follow the intention of the artist more closely but the cost of generating a great quantity of similar models is very high.

Consequently, to automate the process Talton et al. [TLL*11] presented an approach using Markov chain Montecarlo, but its convergence is affected by the fact that the algorithm receives

its feedback from completely-generated models only. Ritchie et al [RMGH15] improved upon it by developing a new sequential Montecarlo algorithm capable of using incremental feedback from incomplete models. Stava et al [SPK*14] developed an inverse procedural modeling algorithm specifically tailored for trees. An input tree is used to estimate the parameters of a recursive algorithm similar to that of [AK84], then new trees resembling the input one may be generated. Still, the needed optimization step is quite expensive.

All previous methods work in 3D, which makes them more powerful when the application demands the actual tree shape (e.g. when adapting a tree to its environment). This comes at a performance cost that we avoid, while maintaining other advantages like authoring ease and the diversity of tree shapes.

3. Overview

We assume a *library* of tree images (RGBA front views of trees) is available. Tree images from the library (or a user-defined subset of it) will be referred to as *exemplars*. Our approach has three stages: exemplar processing, tree synthesis and billboard cloud synthesis.

An overview of the exemplar processing stage is shown in Figure 1. We start by resizing all exemplars to a fixed resolution (Section 4.1). A transparent border is added if necessary to preserve the aspect ratio. Each tree image is segmented into crown and trunk classes using a fine-tuned deep neural network model. If not already provided, an alpha channel could be computed using an alpha matting approach guided by the segmentation (Section 4.3). We then extract all contours of the alpha channel using a border following algorithm, and take the largest external contour as the overall tree contour C (Section 4.2).

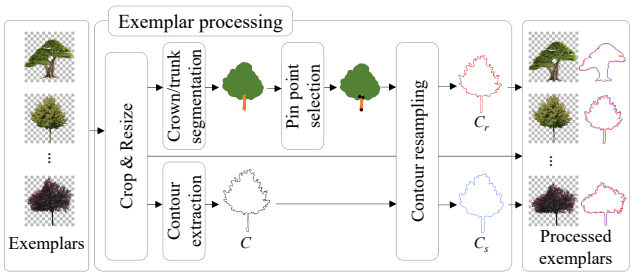

Figure 1: *Overview of the exemplar processing algorithm. Given a collection of RGBA images of trees, we generate fixed-length contours representing the overall shape of the trees.*

Then we re-sample C to create a new contour C_r with a fixed number N of 2D points (Section 4.5). The resampling uses three pin points (tree bottom, trunk top-left, trunk top-right) that are computed from the intersection of the tree contour with the segmented crown/trunk components (Section 4.4).

We then concatenate the (mean-subtracted) (x,y) coordinates of the N contour points onto a $2N$ vector $\boldsymbol{u}$ that represents the overall tree shape in $\mathbb{R}^{2N}$ space. The resampling above guarantees that a fixed range of $\boldsymbol{u}$ components correspond to crown contour points and another fixed range of $\boldsymbol{u}$ components to left/right trunk contour points. We then re-sample again the contour C_r (Section 4.6) to obtain a simplified contour C_s that will be used later to encode interior points of the tree as Mean Value Coordinates.

The output of the preprocessing step is, for each tree exemplar, a couple of contours C_r and C_s, along with a binary mask representing the crown/trunk segmentation.

Regarding single-image tree synthesis (Sect. 5), we propose two variants. The first one creates tree variations from scratch, using exemplars from the library. A new overall contour is created by computing a random convex combination of $\boldsymbol{u}$ vectors and applying a region fill algorithm to handle potential self-intersections in the synthetic contours (Section 5.1). The new contour already defines a preliminary segmentation of the output alpha channel. We use Mean Value Coordinates to morph the exemplar images to the target shape and compute the RGBA color as a combination of these images (Section 5.2). Since we also transfer alpha values, the final shape of the tree is richer than the original contour, i.e. it might include see-through parts within the crown due to sparse foliage on the source images.

The second synthesis variant requires the user to specify a tree image T. This image will be combined with features from the library to create variations of T. In this case, we apply to T all the processing steps we apply to exemplars, and then apply a similar synthesis procedure but giving higher weights to T features.

Each new synthetic tree can be converted into a billboard cloud (Sect. 6). We first *inflate* the tree silhouette to generate a plausible 3D volume of the crown. We then extract compact superpixels from the image, and refine their boundaries to guarantee foliage-like silhouettes. These superpixels are distributed randomly within the reconstructed crown to create the billboard cloud.

4. Exemplar Processing

We now describe the preprocessing steps for each exemplar RGBA image from the library. The outcome of these steps is a suitable encoding of the overall tree shape through a fixed-length contour.

4.1. Image normalization

We start by normalizing the images so that all exemplars have the same size. This normalization is beneficial for subsequent steps, especially for the segmentation through Fully Convolutional Networks (FCN). We first compute a binary version of the alpha channel through alpha thresholding, and set all transparent pixels to black to simplify the classifier task. We then crop the image to the minimum axis-aligned rectangle that encloses all opaque pixels, and add a padding black (and transparent) border to make the image square without distorting the tree. Finally, we resize the image to a fixed size (we used 1024×1024 pixel images for the experiments). Figure 12 shows the normalized versions of some exemplars.

4.2. Contour extraction

We extract the overall (external) tree contour from the alpha channel A. We first threshold the alpha channel (we used 0.5 as threshold, assuming normalized values) to get a binary alpha mask A_t.

Then we apply a border-following algorithm [SA85] to A_t to extract all the contours separating opaque regions from transparent ones. Each contour is represented as a collection of 2D points.

Figure 2: *RGBA images (left), all contours extracted from their alpha channel (middle), and longest external contour (right). Internal contours are shown in light gray, and external contours in black.*

Figure 2 shows the contours extracted from some exemplars. Typical tree images include multiple contours; trees with sparse foliage have multiple see-through parts and even multiple connected components (due e.g. to thin branches not appearing in A_t).

We classify the extracted contours as exterior (not inside any other contour) and interior (inside another enclosing contour). We take as the overall contour the longest exterior contour C (Figure 2). Notice that $C = \{(x_i, y_i)\}$ will have a variable number of points depending on, among other factors, the fractal nature of the tree silhouette.

4.3. Crown/Trunk segmentation

The contour contains both the tree crown as well as the trunk. If not segmented, the new contours extracted through convex combinations could mix straight line segments from the trunk with fractal-like segments from the foliage. Therefore, we need to segment the crown from the trunk. This segmentation will also allow us to generate billboard clouds for foliage and trunk independently (Section 6).

Several image segmentation approaches exist, but in the past few years Deep Learning has been shown to offer remarkable results. We fine tuned the recent state-of-the-art network model DeepLabv3+ [CZP*18] with the Xception-65 backbone [Cho17] using as initialization point the author's provided trained model on the ADE20K dataset [ZZP*16]. We opted for this pre-trained model because that dataset contains more than 7300 images with tree instances, so it is the closest one to our task. We modified the last layer of the network in order to map to three classes: background, crown and trunk. We trained this network for about 80k iterations with 513×513 crops, a batch size of 2, and a low learning rate 10^{-5}.

Regarding the training set, we manually segmented our collection of 95 RGBA tree billboards. In order to add more exemplars and images without alpha matting, we gathered images from ADE20K containing the class trunk or with tree instances annotated with the keyword trunk as long as they covered at least 5% of the image. This search retrieved 164 additional images. While we used them in order to have more exemplars and variety, they are very different from the billboards in our collection: a tree rarely appears isolated and merges with neighboring trees, the segmentation has a polygonal non-detailed shape, and sometimes a tree instance with trunk annotation shows both crown and trunk or, contrarily, clearly visible trunks on the photograph are just labelled as tree. By combining these images and our billboards, we built a training set of 219 images and a test set of 40 images. We also enabled scaling and cropping on the network input as a data augmentation strategy. Note that we did not use a validation split due to the limited number of images and the fact that we are not tuning any network hyper-parameter.

The mean intersection over union obtained on the test set was 67.2%, or 70.1% if we exclude the images from ADE20K which are coarsely segmented. Figure 3 shows segmentation on billboards from this test set as well as three additional photographs. Segmenting an 1024×1024 image takes on average 0.72 s.

For the case of photograph segmentation, the mask is still too coarse for extracting a billboard from it. In this case, we could use this segmentation to automatically create a trimap and compute the alpha channel using image matting approaches. We experimented with the deep convolutional neural network for proposed by Cho et al. [CTK16], using the code and model provided by the authors. Figure 4 shows an example. The trimap grey label is just a dilation kernel over the segmentation, while the white class (alpha 1) is computed as the union of an erosion kernel applied on the trunk mask and random points sampled inside the eroded crown mask.

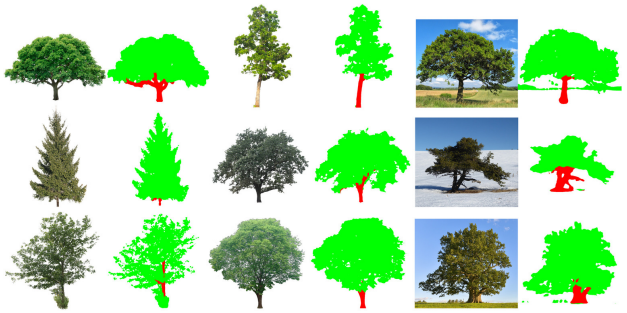

Figure 3: *Billboards and photographs segmented using our fine-tuned DeepLabv3+ model.*

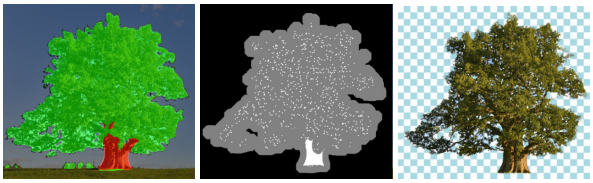

Figure 4: *Segmentation of a tree photograph obtained using our fine-tuned DeepLabv3+ model, automatically derived trimap from this segmentation, and extracted billboard with alpha matting.*

4.4. Pin point selection

The creation of new contours through linear combinations of existing contours requires the definition of a minimum set of matching points across all contours, so that e.g. the first contour point always refers to the point at the tree bottom. For each exemplar, we select three pin points on the extracted contour C: tree bottom (B), trunk top-left (L) and trunk top-right (R), see Figure 5. The tree bottom is set to the point with minimum y. If multiple points share such a minimum, we set B to the median point. For the trunk L and R points, we traverse the points in C, starting from B, in both forward and backward directions. We stop the traversal as soon as the current contour segment intersects the crown baseline, i.e. the lowest height on the segmented crown. In the rare case that the segmented image contains no trunk pixels, we set L=R=B.

4.5. Contour resampling

Extracted contours C are variable-length and thus not suitable for generating variations through linear combinations of exemplars. We thus resample the overall contour C of each exemplar to include exactly N points. Our resampling strategy considers three different segments on C: the crown segment (R to L), the left truck segment (L to B) and the right trunk segment (B to R). Each segment is assigned a fixed number of samples in the resampled contour C_r. We used N =2,000 points, allocating 1,600, 200 and 200 points for each of the three segments above. Resampling within each segment is performed as in chord-length parameterization, i.e. attempting to generate uniform chord lengths between samples. The output contour C_r has thus N points, all of them uniformly distributed (in a chord-length sense) within each segment. Figure 5 shows the resampled contours (in cyan) for a few exemplars.

4.6. Contour simplification

The resampled contour C_r is detailed enough to be used for contour synthesis, but too complex for the generation of Mean Value Coordinates. We thus further resample C_r to $M \ll N$ points to generate a simpler contour C_s. We observed that the distortions on the color image produced by the Mean Value Coordinates after warping the contour are more acute for those pixels near the contour or outside of it. Therefore, we actually generate C_s by first rendering the mask of the interior of C_r, dilating this mask for some iterations (8 in our tests) and resampling to M points the contour of this mask. Figure 5 shows the simplified contours (in red) for a few exemplars and $M = 100$.

The output of the above steps is a collection of processed exemplars along with their resampled C_r and simplified contours C_s.

5. Tree Synthesis

We now discuss different strategies to generate new tree images through random combinations of exemplars. We first define the overall tree shape by synthesizing a new contour through linear combinations of contours (Section 5.1). Then the contour is filled by transferring RGBA color from the exemplars (Section 5.2).

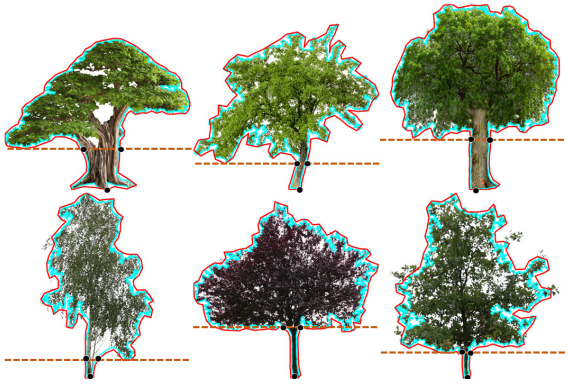

Figure 5: *Resampled contour C_r (in cyan), simplified contour C_s (in red), crown baseline (in orange) and pin points (in black).*

5.1. Contour synthesis

Let $\boldsymbol{u_j}$ be the vector $\in \mathbb{R}^{2N}$ that results from flattening the (x,y) coordinates of the resampled contour C_r from the j-th exemplar.

We can linearly interpolate two contours $\boldsymbol{u'} = (1-t)\boldsymbol{u_0} + t\boldsymbol{u_1}$ with $t \in [0,1]$ to produce a continuous morphing between them. Figure 6 shows several snapshots of the interpolation between two contours. The quality of the interpolated contours is highly dependent on the matching contour points; the use of the B, L, R pin points prevents excessive rotations during morphing.

Figure 6: *Morphing two contours through convex combinations.*

We can extend this idea to incorporate additional contours through convex combinations of existing contours, i.e., $\boldsymbol{u'} = \sum w_i \boldsymbol{u_i}$, with $w_i \geq 0$ and $\sum w_i = 1$. We avoid negative weights to prevent contours from being reflected (e.g. $-\boldsymbol{u_i}$ would result in an upside-down contour).

The generated contour provides a preliminary version of the output alpha channel (to be refined later, see next section). We do this by simply drawing the contour onto a blank alpha channel (with alpha set to 1.0 for all pixels), and then using a region fill algorithm from any seed outside the contour to clear the alpha values of the pixels outside the contour. This method is robust against potential self-intersections of the combined contours.

5.2. Color transfer

In the previous subsections we generated new contours C'_r and their associated alpha masks. We now explain how to fill non-transparent pixels of the output image E' with color.

We address this problem by transferring color from one or more exemplars (*source* images) to the image being synthesized (*target*).

We pose this color transfer problem as an *image warping* problem. Let $w \times h$ be the resolution of the (processed) exemplar images, and let Ω be their $w \times h$ rectangular domain. Given a source contour C_s and a target contour C'_s, both with vertices in Ω, we aim at defining a smooth warp function $f : \Omega \mapsto \Omega$ mapping each vertex $(x_i, y_i) \in C_s$ to the corresponding vertex $(x'_i, y'_i) \in C'_s$. Such a warping function can be used to deform any source image E defined on Ω to a target image E' by simply letting $E' = E \circ f^{-1}$ [HF06].

We define the mapping above through barycentric coordinates with respect to (a simplified version of) the source and target contours. In particular, we use mean value coordinates [HF06], which are well-defined for arbitrary planar polygons.

When transferring RGBA color from a single source exemplar E, the algorithm proceeds as follows. For each non-transparent pixel $p' = (x', y')$ of the target image E', we first compute the mean value coordinates λ'_i of p' with respect to the target contour C'_s. We then find the corresponding point on the source image, $p = f^{-1}(p')$, by simply using the resulting coordinates λ'_i with the vertices $\{v_i\}$ of the chosen source contour C_s, i.e. $p = \sum \lambda'_i v_i$. The final RGBA color for pixel p' is just $E(p)$. As in [HF06], color sampling can be improved through bilinear interpolation on the 2×2 grid of pixels surrounding each source pixel.

The color transfer approach above can be extended to take colors from multiple exemplars. Let (c_j, a_j) be the RGB and A components of the color extracted (through mean value coordinates) from j-th exemplar. We compute the output alpha value as $a' = \max a_j$, i.e. the final pixel will take the highest opacity from the source pixels. We do this to avoid an excessive amount of transparent pixels to be transferred to the target image. The color is computed as a random convex combination of the colors, but considering only those colors with non-zero opacity values. Figure 7 shows the morphing of two exemplars using the RGBA color transfer to fill the interpolated contours.

Figure 7: *Morphing two trees through RGBA color transfer.*

5.3. Histogram transfer

As a result of the previous operations we have been able to generate new trees. We can add more variation by changing the color histogram of the generated image. Since we want the result to be plausible, we use a histogram transfer algorithm [GW07] so that the image we have generated has the same color distribution as another image provided as a reference.

In order to do this we compute the cumulative histograms for both images (source image and template image). We then interpolate linearly to find the unique pixel values in the template image that most closely match the quantiles of the unique pixel values in the source image. This process is performed for each RGB color channel separately and it always ignores transparent pixels.

Histogram matching is also useful both to improve the matching between the combined images, since they can be taken in varying lighting conditions, as well as to simulate the change of vegetation coloration during different seasons. Figure 8 shows an RGB transition using histogram matching.

Figure 8: *Morphing two trees through RGBA color transfer; with histogram transfer (top) and without (bottom).*

6. Billboard cloud synthesis

Billboard clouds are a common representation for volumetric tree foliage. We will now introduce a simple approach for generating a billboard cloud from the segmented tree foliage that will provide more parallax effects and realism on mid-range views than just using the planar photograph.

First, we use SLIC [ASS*12] to partition the photograph into compact superpixels. We empirically found that 256 clusters with compactness 0.35 produce best results in the majority of cases (see Figure 9 for an example). However, some regions – especially in dark areas – are delimited by long rectilinear edges that will produce undesirable perceptual effects when rendering the billboard cloud. Reducing the compactness yields more fractal silhouettes even in these cases, but the different superpixels largely differ in size. Therefore, we propose to replace each straight edge on a region contour by some curve taken from the silhouette of the tree crown following two constraints: the relative positions of the endpoints of the silhouette segment must match with the endpoints of the straight line (up to a small tolerance), and the segment must not cross the straight line. This second constraint was added to ensure that each pixel from the original photograph is represented at least in one region. This way, the frontal view of the billboard cloud will look almost identical to the input photograph. Figure 10 shows some improved contours using this strategy.

Then, we position these billboards to populate the tree crown. Since we want the front view to be as close as possible to the original photograph, we will simply respect their relative positions on the image plane, and compute an extruded depth based on a biharmonic equation with the silhouette contrained to lie on the image plane [ACA16]. Since this equation results in a smooth surface, we improve the positions by deepening the billboards with mean luminance smaller than the overall mean luminance. The reason is that we do not want dark billboards appearing on the silhouette when

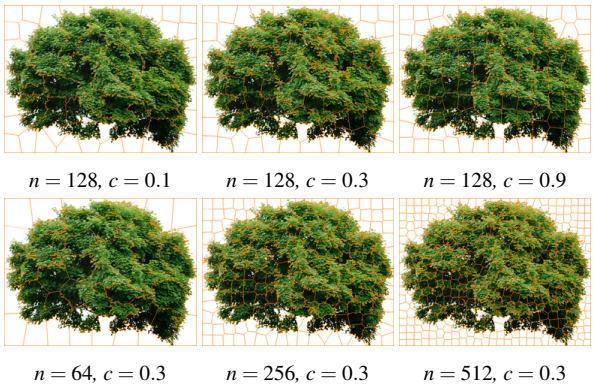

Figure 9: *SLIC superpixels segmentation with varying number of clusters n and compactness c.*

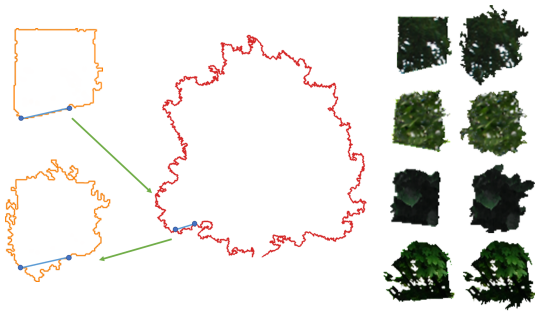

Figure 10: *Improving region segmentation by replacing nearly-straight edges with segments taken from the silhouette of the tree crown. The two columns on the right side show four example superpixels from different trees as extracted from SLIC and after straight edge replacement.*

rendering the tree from lateral views, because we interpret darker areas as more occluded and the billboards on the silhouette do not tend to have occluders, thus producing unnatural views of the tree. By forcing dark billboards to be deeper in the crown, we are also creating cavities and the next layers of billboards will be visible through them, providing convincing parallax effects when rotating the crown. The rear layer of the crown is obtained similarly, but now we horizontally flip the relative positions of the billboards, otherwise we would see a vertical symmetry from a side view. Note that we do not flip the texture but just the position, otherwise we would have a similar issue with the back side being a mirrored version of the front side. Finally, we randomly select billboards and place them at random positions following a blue noise-like distribution inside the crown volume approximated by the frontal and rear layers. We also lower the luminance of these billboards according to their depth inside the crown. The trunk is added as a single billboard, extracted from the picture segmentation. Figure 11 shows some results (see the accompanying video for parallax effects).

Regarding generation times, the SLIC implementation provided in the Python package *scikit-image* plus the improvement of the region contours takes from 20 to 50 seconds depending on the tree. Billboard placement is executed in less than a second for models with up to 500 billboards. Note that this representation can be stored using the tree RGBA picture, the segmentation into crown and trunk, and the contour coordinates of each superpixel.

Figure 11: *Tree image (left) and generated billboard cloud (middle, right). Note how the frontal view looks almost exactly as the original image, and how the deeper billboards appear darker.*

7. Results

Test dataset We tested the preprocessing and synthesis steps using as tree library a collection of 55 RGBA images from different sources, including the VTP Plant Library. The corresponding normalized exemplars are shown in Figure 12.

Segmentations Figure 12 also shows the result of segmenting crown and trunk pixels on the test exemplars, using our fine-tuned Deeplabv3+ model. Notice that the trunk was correctly segmented from the crown in all images where the trunk was visible. Only two images had no visible trunk. For these images, pin points L and R were set to B.

Synthetic contours Figure 13 shows some convex combinations of multiple contour pairs. Due to space limitations, here we only show convex combinations with weights $w_1 = 1/3$, $w_2 = 2/3$ (upper triangle of the table) and $w_1 = 2/3$, $w_2 = 1/3$ (lower triangular part). Despite trying to combine radically-different tree species, most combinations look plausible.

Synthetic trees Figure 14 shows some combinations of multiple tree pairs, distorting the image of each row towards the shape of the image in each column. Again, due to space limitations, here we only show convex combinations with weights $1/3$, $2/3$.

Figure 16 shows more examples obtained by combining two randomly selected exemplars, using a convex combination of the contour $w_1 \in [0.3, 0.7]$ and $w_2 = 1 - w_1$, and setting the color as either the convex combination with the same weights as the contour, or sampling one of the two images directly. Generation time for each of these new trees at 1024×1024 resolution mainly depends on the ratio of non-transparent pixels on the interpolated alpha mask,

Figure 12: *Tree images used as exemplars and their segmentation.*

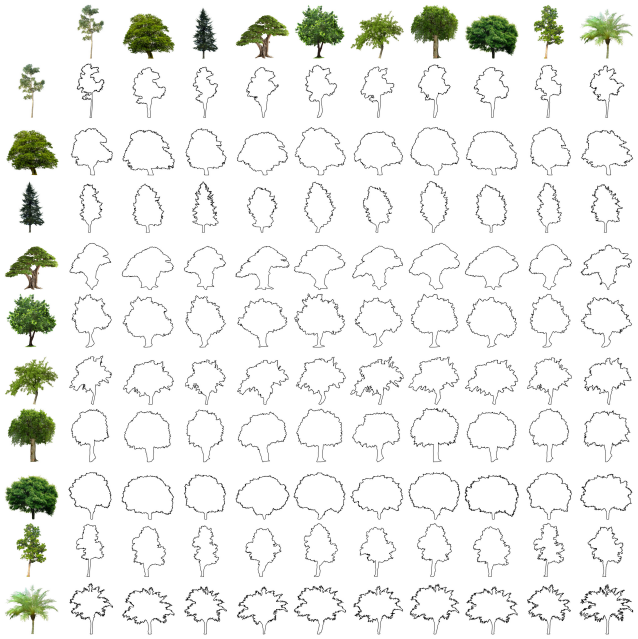

Figure 13: *Contour combination table. For each cell, the contour has been obtained as a combination of the row and column exemplars contours, with corresponding weights* $2/3$ *and* $1/3$.

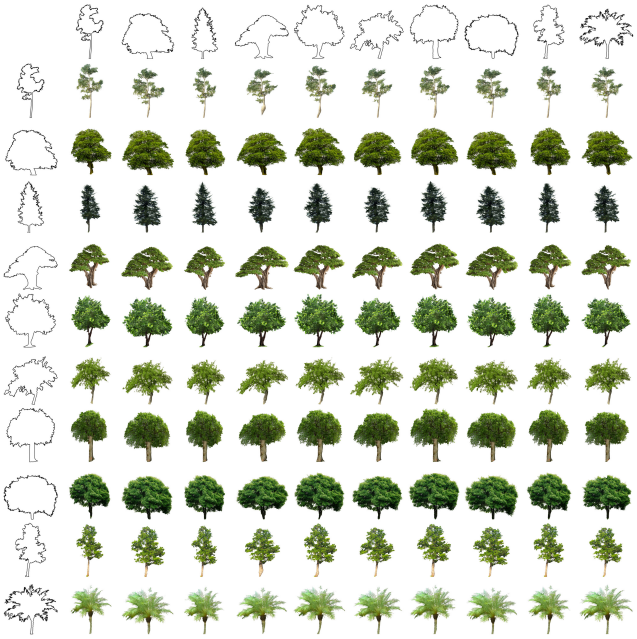

Figure 14: *Tree deformations table. For each cell, the tree image has been distorted to match the shape of the contours of Figure 13.*

as well as on the number M of vertices of the simplified contour C_s used for mean value coordinates. We tried values of M from 100 to 10, and found $M = 20$ to yield very similar visual results to $M = 100$ while reducing the computation time from about 9 seconds to 5.5 seconds per image. Further simplifying C_s to $M = 10$ only reduces the computation to 5 s and causes noticeable distortion artifacts.

Figures 17 and 18 show synthetic trees created by combining exemplars from the same species. Notice how all variations can be recognized as belonging to the corresponding species, and that no identical instances are easily recognizable in Figure 18.

Figure 19 compares images using only the original tree images and our synthetic variations. Despite random size variations, identical tree instances are quite apparent when using a few tree images (top); tree variations (bottom) hinder the detection of similar tree instances.

Although the synthetic tree images we create are not necessarily plausible from a botanical point-of-view (especially when combining exemplars from radically different tree species), these images are still suitable for mainstream applications such as video games, where indeed artists often look for fictional trees.

Limitations The color transfer approach works best when the two contours are not radically different. Otherwise, the source image needs to be severely distorted to fit the target contour, as shown in Figure 15. Similarly, strongly different internal contours might cause crown holes to be filled with unnatural colors.

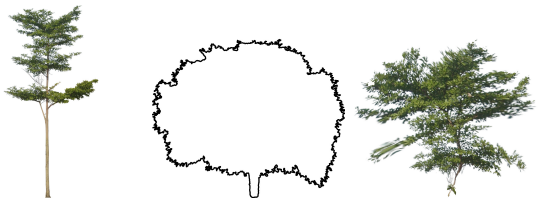

Figure 15: *The thin, sparse tree on the left has been used to fill the thick contour on the middle, resulting in a large distortion.*

8. Conclusions and future work

We have presented an algorithm for the automatic generation of tree variations starting from a collection of example images. The set of exemplars is first used to train a neural network for pixel classification of tree images into canopy and trunk pixels. This allows us to extract pinpoints for the main sections of the tree contour, which facilitates the synthesis of new ones from the exemplars. Synthesis of new tree images is handled by first creating their overall external contour as a random convex combination of existing contours, and then transferring the color from other exemplars through mean value coordinates. The tree variations we generate can be used to author fictional tree images and hybrid specimens, as well as to create variations to prevent tree copies from being spotted. The extracted billboard clouds further increase the applicability in interactive applications. Currently, we focused on crown foliage; we plan to use the FCN segmentation to reconstruct a 3D model of the trunk and its main branches.

Regarding color transfer, Mean Value Coordinates allow multiple polygons as long as the source and target polygon-sets are topologically equivalent. This would allow us to extract multiple

Figure 16: *Randomly generated tree variations from the billboard library. See supplemental material for additional images.*

contours instead of a single outer contour. Since there exist multiple barycentric coordinates generalizations, we would also be interested in analyzing the effect of these on the color transfer step. Another important point is the fact that the provided exemplars may be radically different. While the species are similar (e.g. similar temperate-climate trees) trees match and the distortions are tolerable. This leaves the user with the job of avoiding combinations between completely incompatible trees. A clear improvement of the proposed algorithm would be to compute which subset of trees are sufficiently compatible for their combination, storing this information in a graph to guide the generation of variations.

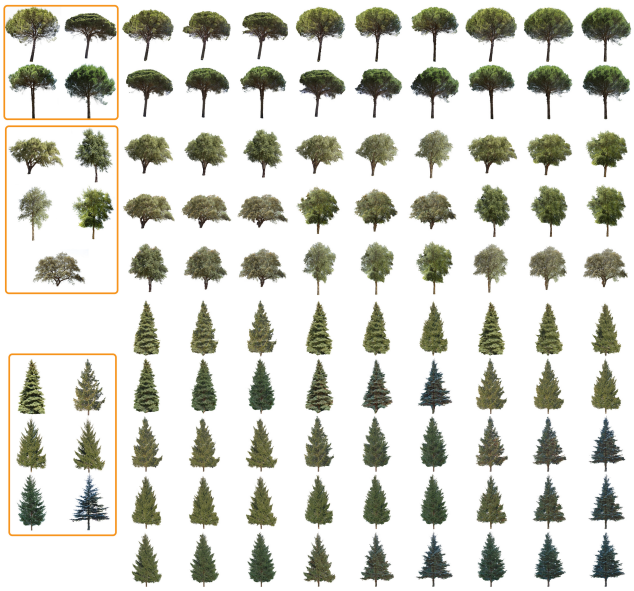

Figure 17: *Variations from 4 pine exemplars (top), 5 oak exemplars (middle) and 6 fir exemplars (bottom).*

Figure 18: *DTM enriched with synthetic trees created by combining exemplars from the same species: 4 pine exemplars (top), and 5 quercus exemplars (bottom).*

References

[ACA16] ARGUDO O., CHICA A., ANDÚJAR C.: Single-picture reconstruction and rendering of trees for plausible vegetation synthesis. Computers & Graphics 57 (2016), 55–67. 1, 2, 6

[ACV*14] ANDÚJAR C., CHICA A., VICO M. A., MOYA S., BRUNET P.: Inexpensive reconstruction and rendering of realistic roadside landscapes. Computer Graphics Forum 33, 6 (2014), 101–117. 1

[AK84] AONO M., KUNII T.: Botanical tree image generation. IEEE Comput. Graph. Appl. 4, 5 (May 1984), 10–34. 3

[ASS*12] ACHANTA R., SHAJI A., SMITH K., LUCCHI A., FUA P., SÜSSTRUNK S.: Slic superpixels compared to state-of-the-art superpixel methods. IEEE Transactions on Pattern Analysis and Machine Intelligence 34, 11 (2012), 2274–2282. 6

[BCF*05] BEHRENDT S., COLDITZ C., FRANZKE O., KOPF J., DEUSSEN O.: Realistic real-time rendering of landscapes using billboard clouds. Computer Graphics Forum 24(3) (2005). 1

[Cho17] CHOLLET F.: Xception: Deep learning with depthwise separable convolutions. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (July 2017), pp. 1800–1807. doi:10.1109/CVPR.2017.195. 4

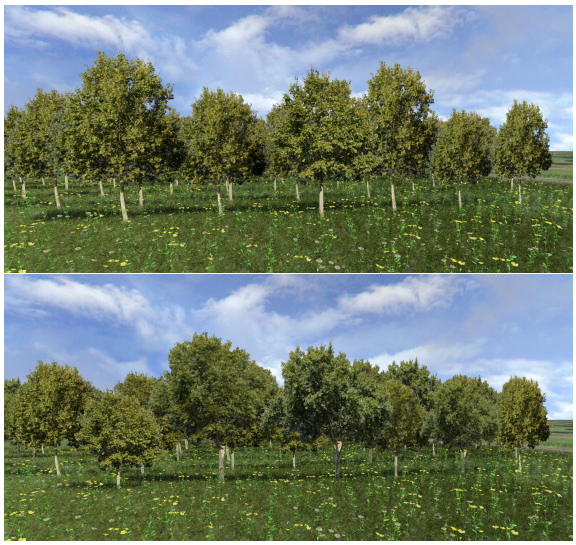

Figure 19: *Scene showing random instances of 3 platanus exemplars without (top) and with (bottom) synthetic variations.*

[CTK16] CHO D., TAI Y.-W., KWEON I.-S.: Natural image matting using deep convolutional neural networks. In ECCV (2016). 4

[CZP*18] CHEN L.-C., ZHU Y., PAPANDREOU G., SCHROFF F., ADAM H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. In ECCV (2018). 4

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In Proceedings of the Conference on Visualization '02 (Washington, DC, USA, 2002), VIS '02, IEEE Computer Society, pp. 219–226. 1

[DN04] DECAUDIN P., NEYRET F.: Rendering forest scenes in real-time. In Rendering Techniques (2004), Keller A., Jensen H. W., (Eds.), Eurographics Association, pp. 93–102. 1

[GMBC13] GUÉNARD J., MORIN G., BOUDON F., CHARVILLAT V.: Reconstructing plants in 3d from a single image using analysis-by-synthesis. In Advances in Visual Computing. Springer, 2013, pp. 322–332. 2

[GW07] GONZALEZ R. C., WOODS R. E.: Digital Image Processing (3rd Edition). Pearson, 2007, pp. 253–260. 6

[HF06] HORMANN K., FLOATER M. S.: Mean value coordinates for arbitrary planar polygons. ACM Trans. Graph. 25, 4 (Oct. 2006), 1424–1441. 2, 6

[KS15] KOHEK Š., STRNAD D.: Interactive synthesis of self-organizing tree models on the gpu. Computing 97, 2 (2015), 145–169. 2

[Lin68] LINDENMAYER A.: Mathematical models for cellular interactions in development: Parts i and ii. Journal of theoretical biology 18, 3 (1968), 280–315. 2

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: Interactive procedural modeling of trees on a tablet. In Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling (2012), SBIM '12, pp. 107–120. 2

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. In Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA, 1996), SIGGRAPH '96, ACM, pp. 397–410. 2

[OOI05] OKABE M., OWADA S., IGARASHI T.: Interactive Design of Botanical Trees using Freehand Sketches and Example-based Editing. Computer Graphics Forum 24, 3 (2005), 487–496. 2

[PHL*] PALUBICKI W., HOREL K., LONGAY S., RUNIONS A., LANE B., MĚCH R., PRUSINKIEWICZ P.: Self-organizing tree models for image synthesis. In ACM SIGGRAPH 2009, pp. 58:1–58:10. 2

[PHM93] PRUSINKIEWICZ P., HAMMEL M. S., MJOLSNESS E.: Animation of plant development. In Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 351–360. 2

[PL96] PRUSINKIEWICZ P., LINDENMAYER A.: The Algorithmic Beauty of Plants. Springer-Verlag New York, Inc., 1996. 2

[PMKL01] PRUSINKIEWICZ P., MÜNDERMANN L., KARWOWSKI R., LANE B.: The use of positional information in the modeling of plants. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 289–300. 2

[PNH*14] PIRK S., NIESE T., HÄDRICH T., BENES B., DEUSSEN O.: Windy trees: Computing stress response for developmental tree models. ACM Trans. Graph. 33, 6 (Nov. 2014), 204:1–204:11. 2

[PSK*12] PIRK S., STAVA O., KRATT J., SAID M. A. M., NEUBERT B., MĚCH R., BENES B., DEUSSEN O.: Plastic trees: Interactive self-adapting botanical tree models. ACM Trans. Graph. 31, 4 (July 2012), 50:1–50:10. 2

[RLP07] RUNIONS A., LANE B., PRUSINKIEWICZ P.: Modeling Trees with a Space Colonization Algorithm. In Proceedings of the Third Eurographics Conference on Natural Phenomena (2007), NPH'07, Eurographics Association, pp. 63–70. 2

[RMGH15] RITCHIE D., MILDENHALL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. ACM Transactions on Graphics (TOG) 34, 4 (2015), 105. 3

[SA85] SUZUKI S., ABE K.: Topological structural analysis of digitized binary images by border following. Computer vision, graphics, and image processing 30(1) (1985), 32–46. 4

[SPK*14] STAVA O., PIRK S., KRATT J., CHEN B., MĚCH R., DEUSSEN O., BENES B.: Inverse procedural modeling of trees. Computer Graphics Forum 33, 6 (2014), 118–131. 3

[TFX*08] TAN P., FANG T., XIAO J., ZHAO P., QUAN L.: Single Image Tree Modeling. ACM Transactions on Graphics 27, 5 (Dec. 2008), 108:1–108:7. 2

[TGY*09] TALTON J. O., GIBSON D., YANG L., HANRAHAN P., KOLTUN V.: Exploratory modeling with collaborative design spaces. In ACM SIGGRAPH Asia 2009 Papers (2009), pp. 167:1–167:10. 2

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. ACM Trans. Graph. 30, 2 (Apr. 2011), 11:1–11:14. 2

[WBCG09] WITHER J., BOUDON F., CANI M.-P., GODIN C.: Structure from silhouettes: a new paradigm for fast sketch-based design of trees. Computer Graphics Forum 28, 2 (2009), 541–550. 2

[XM15] XU L., MOULD D.: Procedural tree modeling with guiding vectors. Computer Graphics Forum 34, 7 (2015), 47–56. 2

[YAMK15] YUMER M. E., ASENTE P., MECH R., KARA L. B.: Procedural modeling using autoencoder networks. In Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (2015), pp. 109–118. 2

[ZZP*16] ZHOU B., ZHAO H., PUIG X., FIDLER S., BARRIUSO A., TORRALBA A.: Semantic understanding of scenes through the ade20k dataset. arXiv preprint arXiv:1608.05442 (2016). 4